ICUFLOWAPP

Code analysis

By: Gateway

2022-03-29

CONTENT

INTRODUCTION

This document contains results of the code analysis of ICUFlowApp.

CONFIGURATION

- Quality Profiles
  - Names: Sonar way [Java]; Stryker [Kotlin]; Stryker [XML];
  - Files: AX9JLBXVZWz9exVyjg9p.json; AX_WRsJAZoNvhpLrXfw9.json; AX_WbTE_ZoNvhpLrXf1b.json;
- Quality Gate
  - Name: Sonar-StrykerAndroid
  - File: Sonar-StrykerAndroid.xml

SYNTHESIS

ANALYSIS STATUS

Reliability	Security	Security Review	Maintainability
A	A	A	A

QUALITY GATE STATUS

Quality Gate Status	Passed

Metric	Value
Reliability Rating on New Code	OK
Security Rating on New Code	OK
Maintainability Rating on New Code	OK
Duplicated Lines (%) on New Code	OK

METRICS

Coverage	Duplication	Comment density	Median number of lines of code per file	Adherence to coding standard
0.0 %	0.5 %	4.1 %	30.0	99.8 %

TESTS

Total	Success Rate	Skipped	Errors	Failures
0	0 %	0	0	0

DETAILED TECHNICAL DEBT

Reliability	Security	Maintainability	Total
-	-	0d 0h 45min	**0d 0h 45min**

METRICS RANGE

	Cyclomatic Complexity	Cognitive Complexity	Lines of code per file	Comment density (%)	Coverage	Duplication (%)
Min	0.0	0.0	4.0	0.0	0.0	0.0
Max	383.0	346.0	5507.0	81.0	0.0	12.8

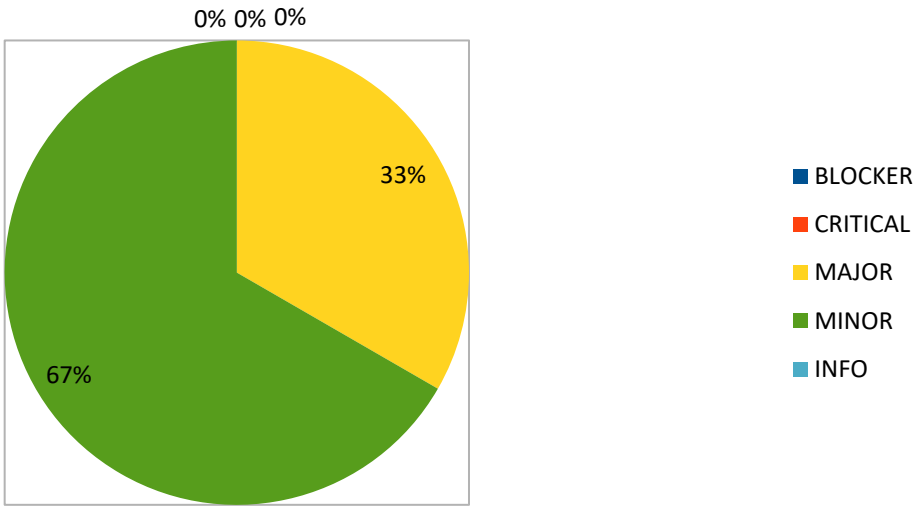
VOLUME

Language	Number
Java	206
Kotlin	3064
XML	2237
Total	5507

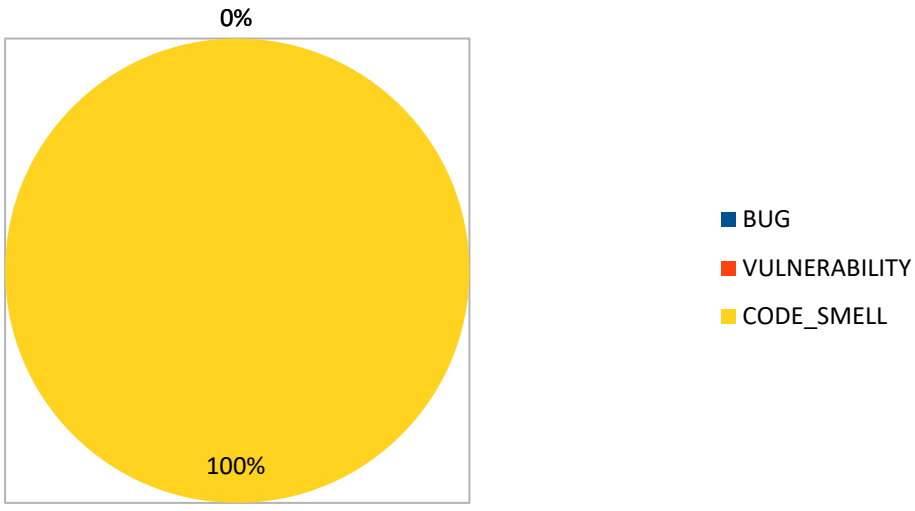
ISSUES

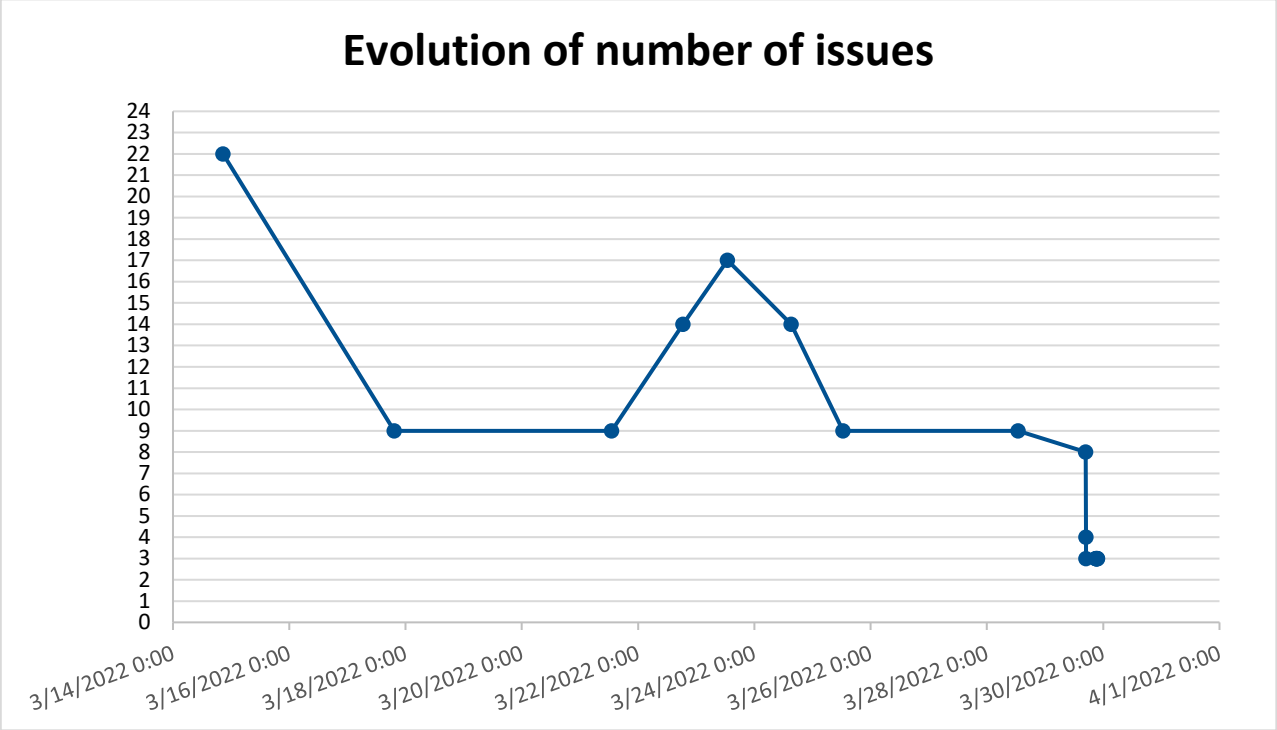
CHARTS

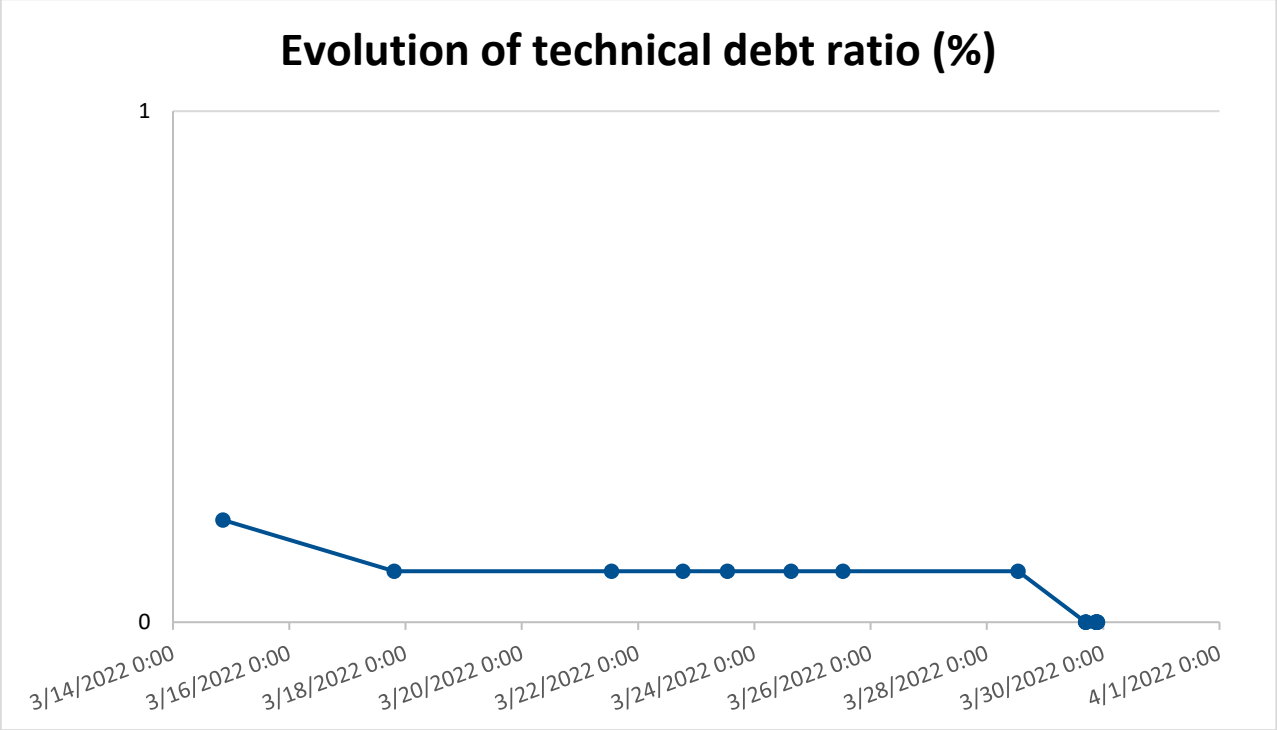
Number of issues by severity



Number of issues by type




Evolution of number of issues
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
3/14/2022 0:00
3/16/2022 0:00
3/18/2022 0:00
3/20/2022 0:00
3/22/2022 0:00
3/24/2022 0:00
3/26/2022 0:00
3/28/2022 0:00
3/30/2022 0:00
4/1/2022 0:00


Evolution of technical debt ratio (%)
1
0
3/14/2022 0:00
3/16/2022 0:00
3/18/2022 0:00
3/20/2022 0:00
3/22/2022 0:00
3/24/2022 0:00
3/26/2022 0:00
3/28/2022 0:00
3/30/2022 0:00
4/1/2022 0:00

ISSUES COUNT BY SEVERITY AND TYPE

Type / Severity	INFO	MINOR	MAJOR	CRITICAL	BLOCKER
BUG	0	0	0	0	0
VULNERABILITY	0	0	0	0	0
CODE_SMELL	0	2	1	0	0

ISSUES LIST

Name	Description	Type	Severity	Number
Methods should not have too many parameters	A long parameter list can indicate that a new structure should be created to wrap the numerous parameters or that the function is doing too many things. Noncompliant Code Example With a maximum number of 4 parameters: public void doSomething(int param1, int param2, int param3, String param4, long param5) { ... } Compliant Solution public void doSomething(int param1, int param2, int param3, String param4) { ... } Exceptions Methods annotated with : Spring's @RequestMapping (and related shortcut annotations, like @GetRequest) JAX-RS API annotations (like @javax.ws.rs.GET) Bean constructor injection with @org.springframework.beans.factory.annotation.Autowired CDI constructor injection with @javax.inject.Inject @com.fasterxml.jackson.annotation.JsonCreator may have a lot of parameters, encapsulation being possible. Such methods are therefore ignored.	CODE_SMELL	MAJOR	1
Class variable fields should not have public accessibility	Public class variable fields do not respect the encapsulation principle and has three main disadvantages: Additional behavior such as validation cannot be added. The internal representation is exposed, and cannot be changed afterwards. Member values are subject to change from anywhere in the code and may not meet the programmer's assumptions. By using private attributes and accessor methods (set and get), unauthorized modifications are prevented. Noncompliant Code Example public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked public String firstName; // Noncompliant } Compliant Solution public class MyClass { public static final int SOME_CONSTANT = 0; // Compliant - constants are not checked private String firstName; // Compliant public String getFirstName() { return firstName; } public void setFirstName(String firstName) { this.firstName = firstName; } } Exceptions Because they are not modifiable, this rule ignores	CODE_SMELL	MINOR	1

	public final fields. Also, annotated fields, whatever the annotation(s) will be ignored, as annotations are often used by injection frameworks, which in exchange require having public fields. See MITRE, CWE-493 - Critical Public Variable Without Final Modifier			
Code annotated as deprecated should not be used	Code annotated as deprecated should not be used since it will be removed sooner or later. Noncompliant Code Example @Deprecated("") interface Old class Example : Old // Noncompliant	CODE_SMELL	MINOR	1

SECURITY HOTSPOTS

SECURITY HOTSPOTS COUNT BY CATEGORY AND PRIORITY

Category / Priority	LOW	MEDIUM	HIGH
LDAP Injection	0	0	0
Object Injection	0	0	0
Server-Side Request Forgery (SSRF)	0	0	0
XML External Entity (XXE)	0	0	0
Insecure Configuration	0	0	0
XPath Injection	0	0	0
Authentication	0	0	0
Weak Cryptography	0	0	0
Denial of Service (DoS)	0	0	0
Log Injection	0	0	0
Cross-Site Request Forgery (CSRF)	0	0	0
Open Redirect	0	0	0
SQL Injection	0	0	0
Buffer Overflow	0	0	0
File Manipulation	0	0	0
Code Injection (RCE)	0	0	0
Cross-Site Scripting (XSS)	0	0	0
Command Injection	0	0	0
Path Traversal Injection	0	0	0

HTTP Response Splitting	0	0	0
Others	0	0	0

SECURITY HOTSPOTS LIST